**Page Object Model | POM**

Creating Selenium test cases can result in an unmaintainable project. One of the reasons is that too many duplicated code is used. Duplicated code could be caused by duplicated functionality and this will result in duplicated usage of locators. The disadvantage of duplicated code is that the project is less maintainable. If some locator will change, you have to walk through the whole test code to adjust locators where necessary. By using the page object model we can make non-brittle test code and reduce or eliminate duplicate test code. Beside of that it improves the readability and allows us to create interactive documentation. Last but not least, we can create tests with less keystroke. An implementation of the page object model can be achieved by separating the abstraction of the test object and the test scripts.

**Note:**We will follow the same example which we have used in ***Login Test Case***.

* *Launch the Firefox browser*
* *Open website*
* *Print a Message to display that the website is opened successfully*
* *Enter username , password and click login button*
* *Print a Message to display that the login successfully*
* *Wait for 5 Seconds*
* *Close the Browser*

**package com.smita.webdriver.locator;**

**import org.openqa.selenium.By;**

**import org.openqa.selenium.WebDriver;**

**import com.smita.webdriver.util.ChromeDriverUtil;**

**/\*\*\* @author Smita\*\*/**

**public class TwitterLoginTest {**

**public static void main(String[] args) throws InterruptedException {**

**System.*out*.println(" get the driver");**

**WebDriver driver = ChromeDriverUtil.*getDriver*();**

**driver.get("https://twitter.com/login");**

**System.*out*.println(" lets load the website with get method");**

**driver.findElement(By.*xpath*("//\*[@id=\"page-container\"]/div/div[1]/form/fieldset/div[1]/input")).sendKeys("smitaselenium3");**

**driver.findElement(By.*xpath*("//\*[@id=\"page-container\"]/div/div[1]/form/fieldset/div[2]/input")).sendKeys("Password@123");**

**Thread.*sleep*(2000);**

**driver.findElement(By.*xpath*("//\*[@id=\"page-container\"]/div/div[1]/form/div[2]/button")).click();**

**Thread.*sleep*(2000);**

**String url = driver.getCurrentUrl();**

**System.*out*.println("Current URL : "+url);**

**if(url.equals("https://twitter.com/")) {**

**System.*out*.println("Login Passed");**

**}**

**else**

**System.*err*.println("Login Failed");**

**}**

**}**

***Old TwitterLogin Test Case***

Let’s assume it our base test case and implement the Page Object Model (POM) in it.

**How to do it…**

1. Create a ‘***New Package*‘** file and name it as ‘**com.smita.selelium.pageObjects’**, by right click on the Project and select **New** > **Package**. We will be creating different packages for Page Objects, Utilities, Test Data, Test Cases and Modular actions. It is always recommended to use this structure, as it is easy to understand, easy to use and easy to maintain.

2. Create a ***‘New Class*‘** file and refer the name to the actual page from the test object, by right click on the above created Package and select **New** > **Class**. In our case it is **LogIn** **Page**.

3. Now create a **Static Method** for each **Element** (Object) in the Home Page. Each method will have an **Argument** (driver) and a **Return** value (element).

**Driver** is being passed as an **Argument** so that Selenium is able to locate the element on the browser (driver).

**Element** is returned, so that an **Action** can be performed on it.

**Method** is declared as **Public Static**, so that it can be called in any other method without **instantiate** the class.

Follow the rule for creating **LogIn Page** class.

|  |  |
| --- | --- |
|  | **package com.smita.selelium.pageObjects;**  **import org.openqa.selenium.By;**  **import org.openqa.selenium.WebDriver;**  **import org.openqa.selenium.WebElement;**  **/\*\*\* @author Smita\*\*/**  **public class LogInPom {**  **private static WebElement *element* = null;**  **//method for username input element**  **public static WebElement txtbx\_UserName(WebDriver driver) {**  ***element* = driver.findElement(By.*xpath*("//\*[@id=\"page-container\"]/div/div[1]/form/fieldset/div[1]/input"));**  **return *element*;**  **}**  **//method for password input element**  **public static WebElement txtbx\_Password(WebDriver driver) {**  ***element* = driver.findElement(By.*xpath*("//\*[@id=\"page-container\"]/div/div[1]/form/fieldset/div[2]/input"));**  **return *element*;**  **}**  **//method for logIn button**  **public static WebElement btn\_LogIn(WebDriver driver) {**  ***element* = driver.findElement(By.*xpath*("//\*[@id=\"page-container\"]/div/div[1]/form/div[2]/button"));**  **return *element*;**  **}**  **}** |

4) Create a ‘[***New Class***](http://toolsqa.com/selenium-webdriver/configure-eclipse-with-selenium-webdriver/)‘ and name it as **POM\_TC** by right click on the ‘**automationFramework**‘ Package and select **New** > **Class**. We will be creating all our test cases under this package.

Now convert your old ***Login Test Case*** in to the new Page Object Model test case.

|  |  |
| --- | --- |
|  | **package com.smita.selelium.pageObjects;**  **import java.util.concurrent.TimeUnit;**  **import org.openqa.selenium.WebDriver;**  **import com.smita.webdriver.util.ChromeDriverUtil;**  **/\*\*\* @author Smita\*\*/**  **public class PageObjectModel {**  **private static WebDriver *driver* = null;**  **public static void main(String[] args) throws InterruptedException {**  ***driver* = ChromeDriverUtil.*getDriver*();**  ***driver*.manage().timeouts().implicitlyWait(10, TimeUnit.*SECONDS*);**  ***driver*.get("https://twitter.com/login");**  **// Use page Object library now**  **LogInPom.*txtbx\_UserName*(*driver*).sendKeys("smitaselenium3");**  **LogInPom.*txtbx\_Password*(*driver*).sendKeys("Password@123");**  **LogInPom.*btn\_LogIn*(*driver*).click();**  **System.*out*.println(" Login Successfully, now it is the time to Log Off buddy.");**  **Thread.*sleep*(5000);**  ***driver*.quit();**  **}**  **}** |

You will notice that once you type ***LogInPom*** in your test script and the moment you press dot, all the methods in the ***LogInPom*** will display. We can expose methods in order to reduce duplicated code. We are able to call these method multiple times. This will ensure a better maintainable test code, because we only have to make adjustments and improvements in one particular place.

Your Project explorer window will look like this now.

